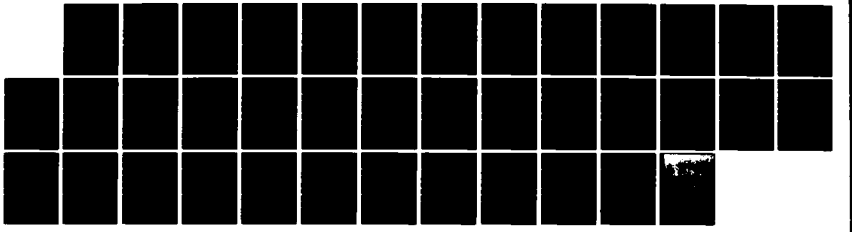
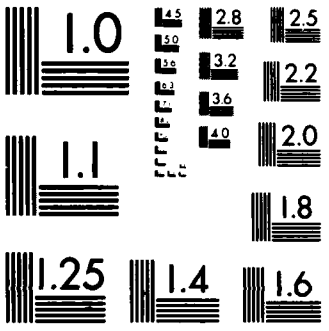
AD-A132 874 TREGRM-83 REPORT AND USER'S MANUAL(U) COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE G M CLEMM JAN 83 CU-CS-249-83 ARO-15074.15-MA DAAG29-78-G-0046 1/1

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A132874

(12)

UNIVERSITY OF COLORADO

TREGRM-83 Report and User's Manual

by

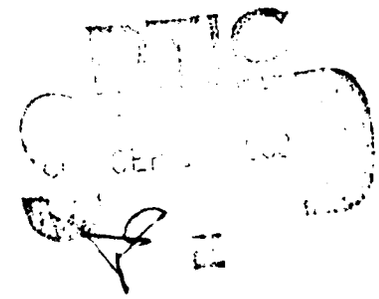
Geoffrey M. Clemm
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

CU-CS-249-83 January 1983

DTIC FILE COPY

DEPARTMENT OF COMPUTER SCIENCE
CAMPUS BOX 430
UNIVERSITY OF COLORADO, BOULDER
BOULDER, COLORADO 80309

Technical Report



This document has been approved for public release and sale; its distribution is unlimited.

12

TREGRM-83 Report and User's Manual

by

Geoffrey M. Clemm
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

CU-CS-249-83 January 1983

INTERIM TECHNICAL REPORT
U. S. ARMY RESEARCH OFFICE
CONTRACT NO. DAAG29-78-G-0046

Approved for public release;
Distribution Unlimited

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

We acknowledge U. S. Army Research support under contract no. DAAG29-78-G-0046 and National Science Foundation support under grant no. MCS77-02194

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 15074.15-MA	2. GOVT ACCESSION NO.: AD-A132 574
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): TREGRM-83 Report and User's Manual	5. TYPE OF REPORT & PERIOD COVERED: Technical
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Geoffrey M. Clemm	8. CONTRACT OR GRANT NUMBER(s): DAAG29 78 G 0046
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Colorado, Boulder, CO 80309	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: Jan 83
	13. NUMBER OF PAGES: 35
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

programming languages
computer programs
compilers
object code

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

TREGRM is a language for specifying the creation of parse trees for any current programming language, including FORTRAN. This report describes the TREGRM language, a compiler for the language, and an interpreter for the resulting object code. The interpreted object code, in conjunction with a lexical analyzer, forms a syntactic analyzer that takes as input a stream of characters and produces as output a parse tree. The compiler and interpreter are designed for portability. Both are written in ANSI FORTRAN (1966) supplemented by a small number of short machine dependent subroutines. Included is a TREGRM program describing the syntax of FORTRAN-77.

CONTENTS

INTRODUCTION 1

THE LANGUAGE 2

Lexical Analyzer Interface 3

Reserved Word Specification 4

Node Names 5

Rules 6

TREGRM Regular Expressions 8

Tree-Building Actions 10

THE COMPILER 11

USING THE COMPILER 13

THE STANDARD DRIVER 14

Appendix A : Compiler machine dependencies 18

Appendix B : Standard Driver Machine Dependencies 20

Appendix C : Examples of TREGRM Programs 21

TREGRM 22

FORTRAN-77 24

Abstract

TREGRM is a language for specifying the creation of parse trees for any current programming language, including FORTRAN. This report describes the TREGRM language, a compiler for the language, and an interpreter for the resulting object code. The interpreted object code, in conjunction with a lexical analyzer, forms a syntactic analyzer that takes as input a stream of characters and produces as output a parse tree. The compiler and interpreter are designed for portability. Both are written in ANSI FORTRAN ~~(1966)~~ supplemented by a small number of short machine dependent subroutines. Included is a TREGRM program describing the syntax of FORTRAN-77.

1. INTRODUCTION

The first phase of the analysis of a computer program is lexical analysis, where the source text is broken up into the words or "tokens" of the programming language. The second phase is "syntactic analysis" or "parsing", where structure of the program is determined.

The result of parsing is often a sequence of calls to a semantic routine with a "reduction number" or "action number" as the argument. To provide a more structured interface between the syntactic analysis and semantic analysis of a program, a parse tree can be created, where the parse tree is a "flattened" (simplified) version of the derivation tree for that program. To support the production of parse tree generators, the TREGRM System was developed. The TREGRM system consists of a language, a compiler for the language, a parser generator, and an interpreter for the object code produced by the TREGRM compiler and the parser generator. The TREGRM compiler was designed to allow easy retargeting to different lexical analyzer and parser generators. Currently, the compiler expects the lexical analyzer to be one produced by the FSCAN [1] system, and the parser generator to be one produced by the LR parser generating system [2]. Versions of the compiler that target the CLEMSW [3] and YACC[4] parser generating systems are also available.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Codes
/or
A

2. THE LANGUAGE

The TREGRM language (henceforth referred to simply as "TREGRM") was designed to allow the specification of a complex parse tree generator in as concise and understandable a manner as possible.

A TREGRM program consists of three sections, with an optional fourth section. The required sections specify the interface to the lexical analyzer, the node names for the parse tree, and the rules for building the parse tree. The optional section specifies reserved words that are to be screened out of tokens from the lexical analyzer, and is placed following the lexical analyzer interface section.

2.1. Lexical Analyzer Interface

Syntax

The lexical analyzer interface section consists of the keyword, 'SCANNER', followed by a sequence of token translation rules. Each token translation rule contains a TREGRM terminal name and an FSCAN token name, separated by an equals sign (=). The FSCAN token name is the name of a token as specified in the 'TOKENS' section of an FSCAN program, without any enclosing quote marks. The TREGRM terminal name is a sequence of characters enclosed in either single or double quotes. Single (double) quotes specify the terminal to be a "deleted-terminal" ("kept-terminal"). A kept-terminal translation rule also contains a node name, which follows the FSCAN token name, separated by a double arrow (=>).

Example

```
SCANNER
    "HOLLERITH CONSTANT" = HCONST => "HCNODE"
    "VARIABLE NAME" = NAME => "NMNODE"
    '(' = LPAREN
    ')' = RPAREN
```

Semantics

The token translation rules specify the mapping from the tokens, as produced by the lexical analyzer specified by the FSCAN program, to the terminals, as used in the TREGRM program. There must be one token translation rule for each terminal used in the parse tree rules of the TREGRM program. Some of the tokens in the token stream are "variable" tokens, such as tokens of type "identifier" or "integer" (as opposed to tokens of type "equals sign" or "if-keyword"). These tokens contain a sequence of characters that are the "value" of that token. These values must be preserved in the parse tree if the parse tree is to contain the information needed for most semantic processing. To store these values, leaf nodes are created for each variable token, where the leaf node contains a pointer to the appropriate character sequence value. Such variable tokens are indicated as kept-terminals in a kept-terminal token translation rule. The node name of the leaf for tokens of that type is specified in the kept-terminal token translation rule.

2.2. Reserved Word Specification

When developing a new language, it is often the case that the lexical analysis can be specified satisfactorily very early in in the development process, except for the exact set of reserved words, operators, and delimiters to be recognized. To support this, TREGRM provides an optional section where these reserved words can be specified.

Syntax

The reserved word specification section consists of the keyword, 'SCREENER', followed by a sequence of deleted-terminals enclosed in parentheses, and a sequence of token names also enclosed in parentheses. The two sequences are separated by an equals sign.

Example

```
SCREENER
   ('BEGIN' 'END' 'PROGRAM' ':=' '+') = (NAME OPRATR)
```

Semantics

At runtime, the value of each token whose token name is one of those specified in the SCREENER section is examined before it is translated according to the token translation rules. If the sequence of characters is identical to the sequence of characters in the name of a deleted-terminal in the SCREENER section, the token will be translated to be that deleted-terminal, rather than the terminal specified in the token translation rule for that token name.

2.3. Node Names

Syntax

The node name section consists of the keyword, 'NODES', followed by a sequence of node names, where each node name consists of a sequence of characters enclosed in double quotes. To allow the use of node names directly in FORTRAN programs using the parse tree produced, the sequence of characters should be a legal FORTRAN identifier.

Example

```
NODES
   "NAMEND" "INTGND" "LABLND"
   "PLUSND" "EQULND"
```

Semantics

All node names that will be used in the parse tree must be declared in the node name section. This includes the names of leaf nodes specified in the token translation rules. The name of a node will be indicated in the parse tree by a positive integer, where the i'th node name in the node name list will be assigned the number, i. One of the tables produced by the TREGRM compiler is a FORTRAN block data containing a common block whose elements are the node names as declared in the node name section. These variables are initialized to the appropriate values in the block data subprogram.

2.4. Rules

Syntax

The rules section consist of the keyword, 'RULES', followed by a sequence of TREGRM rules, where each rule is terminated by a semicolon. As in a BNF rule, the left side of a TREGRM rule is a nonterminal while the right side is a sequence of alternatives. Each alternative may have an associated tree-building action, and an alternative, rather than being only a sequence of terminals and nonterminals, may contain any of a variety of operators, in the style of regular expressions, as well as parentheses for grouping. Each alternative is preceded by a single-right-arrow (->). The optional tree-building action is placed at the end of the corresponding alternative and is preceded by a double-right-arrow (=>).

Example

```
RULES
   PROGRAM
      -> tregrm_reg_exprn_1 => action_1
      -> tregrm_reg_exprn_2
      -> tregrm_reg_exprn_3 => action_2 ;
   STATEMENT
      -> tregrm_reg_exprn_4 => action_3 ;
```

Semantics

The nonterminal of the first rule in the rules section is the goal symbol of the grammar. Each TREGRM rule is expanded into a sequence of one or more standard BNF rules. These BNF rules are generated in a format appropriate to the expected input of a parser generator. In addition, a table of tree building actions is generated, with one action for each possible reduction in the generated BNF grammar. During execution of the generated parser, each reduction causes an ordered sequence of zero or more nodes to be associated with the nonterminal corresponding to the alternative being reduced. The result of the parsing is the the sequence of nodes associated with the nonterminal that is the goal symbol of the grammar.

If there is no tree-building action for a given alternative, the sequence of nodes associated with the nonterminal for that alternative is simply the ordered concatenation of the nodes associated with the sequence of nonterminals that make up the right side of the alternative.

If there is a tree-building action for a given alternative, a new node is generated whose name is the node name specified in the tree-building action, and whose sons

are the ordered concatenation described above. The node associated with the nonterminal of the alternative is the newly generated node.

2.4.1. TREGRM Regular Expressions

2.4.1.1. Atomic units

The atomic units of an TREGRM regular expression are terminals and nonterminals.

2.4.1.1.1. Terminals

Syntax

A terminal is either a "kept-terminal" or a "deleted-terminal." A kept-terminal is a sequence of characters enclosed in double quotes (") while a deleted-terminal is a sequence of characters enclosed in single quotes ('). If a sharp (#) appears in the string, the sharp is ignored and the immediately following character is treated as the next character of the string, even if that character is a quote or a sharp.

Examples

```
"NAME" '(' ':=' "INTEGER"
```

Semantics

A deleted-terminal simply indicates a terminal symbol that is to occur in the generated BNF grammar. A kept-terminal is replaced by a generated nonterminal. This nonterminal always causes a tree-building action to occur which generates a leaf whose node name is as specified in the token translation rule for that kept-terminal.

2.4.1.1.2. Nonterminals

Syntax

A nonterminal is a sequence of letters and digits, the first of which is a letter.

Examples

```
A  TEMP  TEMP1  B3B
```

Semantics

A nonterminal becomes one of the nonterminals in the generated BNF grammar.

2.4.1.2. Operations

Syntax

Let A, B, and C be TREGRM regular expressions.

Concatenation	:	A B C . . .
Parenthesization	:	(A)
Repetition	:	A+
List Repetition	:	A // B

Example

```
"NAME" '=' ("INTEGER" // ',')
```

Semantics

A concatenation is mapped directly into the BNF rule.

A parenthesized expression of the form, "(reg_exp)", is replaced by a generated nonterminal, "dummy", where dummy is defined as :

```
dummyA -> reg_exp ;
```

A repetition, "A+", is replaced by a generated nonterminal, "dummyA", where dummyA is defined as :

```
dummyA
   -> A
   -> dummyA A ;
```

A list repetition, "A // B", is replaced by a generated nonterminal, "dummyA", where dummyA is defined as :

```
dummyA
   -> A
   -> dummyA B A ;
```

2.4.2. Tree-Building Actions

Syntax

A tree-building action is a node name, optionally surrounded by parentheses or followed by a question mark.

Examples

```
"NAMEND"  "PLUSND"?   ("SUBRND")
```

Semantics

The node name specifies the name of the node that is to be generated when the corresponding alternative in the BNF grammar is reduced.

If the node name is followed by a question mark, the node is not to be generated if it would receive exactly one son. This action is useful if it is desireable that the common nesting of "expression", "term", "factor", and "primary" be flattened out of the tree whenever possible.

If the node name is to be enclosed in parentheses, the node is generated, and then the subtree rooted at that node is written out to a data file, and the subtree is replaced in primary memory by a single node with a flag indicating that the sons of the node have been written out to the data file. This action is useful if a sequence of parse trees for the logical units of the program are desired, or if the entire parse tree would not fit in available primary memory. The logical unit numbers of the files to which the parse trees are to be sent can be set by a call of the form :

```
CALL TRETAB (SYMTAB,  PRSTAB)
```

where SYMTAB and PRSTAB are integer variables. Two logical unit numbers are specified since it is feasible to send the parse trees and the corresponding symbol tables to different files, although this would not usually be done.

3. THE COMPILER

The TREGRM compiler consists of 3500 lines of standard ANSI FORTRAN code. In addition, there is a group of short (1 to 5 lines) routines that are machine dependent. (See Appendix A).

The compiler takes two input files - a TREGRM program and the tables produced by the FSCAN compiler , and produces five output files - a listing file annotated with the number of the first token on each line, a file containing the tables for driving the lexical analyzer interface and the tree builder, a file containing the generated BNF grammar, a file containing tables specifying the node name to internal integer mapping, and an errors file describing any errors in the input. The files are associated with the FORTRAN logical unit numbers five, seven, six, eight, nine, ten, and zero respectively.

The compiler contains six processing modules that perform the following tasks:

3.1. Lexical Analysis, Syntactic Analysis, and Tree Construction

The input is read and all syntactic errors are reported. If the input is syntactically correct, a parse tree corresponding to the input grammar is built, otherwise processing stops after the entire input has been scanned for syntactic correctness.

3.2. Lexical Interface Verification

The following errors are detected and reported:

(1) A token translation rule for the token type has already been specified.

(2) The token type specified in a token translation rule does not occur in the FSCAN program.

If any of the above errors occur, processing is halted following the completion of the lexical interface verification phase.

3.3. Generation of the BNF Grammar

The following errors are detected and reported:

(3) A node name is used but not declared.

(4) A terminal is used as a deleted-terminal, but was defined as a kept-terminal.

(5) A terminal is used as a kept-terminal, but was defined as a deleted-terminal.

(6) The rule is too complex, rewrite with fewer operators.

(7) The terminal was declared, but not used.

(8) The nonterminal was used in the right hand side of a rule, but not declared in the left hand side of any rule.

If any of the above errors occur, processing is halted following the completion of the BNF generation phase.

3.4. Generation of Lexical Interface Tables

3.5. Generation of Tree-Building Tables

3.6. Generation of Node Name Mapping Tables

4. USING THE COMPILER

With the standard version of the TREGRM compiler, the lexical analysis is specified by an FSCAN program and the parser generator is the LR compiler. The FSCAN compiler and LR compiler read in from logical unit 5, write a listing file to logical unit 6, write the generated tables to logical unit 7, and write error messages to logical unit 0.

With this version, a parse-tree generator can be produced as follows :
(assume that the FSCAN program is named 'lang.fsi' and that the TREGRM program is named 'lang.tgi')

```
ASSIGN lang.fsi Channel_5
ASSIGN lang.fsl Channel_6
ASSIGN lang.fst.f Channel_7
ASSIGN lang.fse Channel_0
RUN fscan_compiler
ASSIGN lang.tgi Channel_5
ASSIGN lang.fst.f Channel_7
ASSIGN lang.tgl Channel_6
ASSIGN lang.tgt.f Channel_8
ASSIGN lang.pgi Channel_9
ASSIGN lang.tgt2.f Channel_10
ASSIGN lang.tge Channel_0
RUN tregrm_compiler
ASSIGN lang.pgi Channel_5
ASSIGN lang.pgl Channel_6
ASSIGN lang.pgt.f Channel_7
ASSIGN lang.pge Channel_0
RUN lr_compiler
```

The four FORTRAN source files, lang.fst.f, lang.tgt.f, lang.pgt.f, and lang.tgt2.f, combined with a standard driver, will generate a parse tree from an input stream of source text.

5. THE STANDARD DRIVER

The standard driver is invoked by a single call of the form

```
CALL PARSER (ITREE)
```

where ITREE is a result parameter pointing to the root of the generated parse tree. The parse tree consists of a set of nodes. Every node has a name (one of the names specified in the "node names" section of the tregrm program specifying that parse tree). Every parse tree node is related to an ordered set of zero or more parse tree nodes, which are called the "sons" of the node. Every node is the son of some other node, except for the root which is the son of no node. A node that has no sons is called a "leaf". Each leaf may contain a "symbol", which is a string of characters obtained from the source text for which the parse tree was built.

An example of a parse tree would be a set of three nodes, Node1, Node2, and Node3. The source text from which the parse tree was built is :

```
XVAL = 134
```

The *tregrm program* used to specify the parse tree is :

```
SCANNER
   "Variable" = SCNNAM => "NAME"
   "Integer" = SCNINT => "INTEGR"
   '=' = SCNEQL
NODES
   "ASSIGN" "NAME" "INTEGR"
RULES
   AssignStatement -> "Variable" '=' "Integer" => "ASSIGN" ;
```

The names of Node1, Node2, and Node3 are ASSIGN, NAME, and INTEGR, respectively. The root of the parse tree is Node1. Node1 has two sons - the first son is Node1 and the second son is Node2. Node2 and Node3 have no sons, and therefore are leaves. The symbol contained by Node2 is "XVAL" and the symbol contained by Node3 is "134".

Two parameterless integer functions are available to determine how many errors occurred during parsing :

```
INTEGER FUNCTION GTRERR ()
INTEGER FUNCTION GTFERR ()
```

where GTRERR returns the number of recoverable errors (parse tree was built), and GTFERR returns the number of fatal errors (parse tree could not be built).

The following functions are available for accessing the generated parse tree and the symbols associated with leaves of the parse tree :

```
INTEGER FUNCTION PTNMSN (NODE)

Mnemonic :
   Parse-tree-node number of sons.
Input Parameters :
   NODE(integer) - a parse tree node.
Result :
   The number of sons of the node, NODE.
   If NODE is not a valid node, -1 is returned.


INTEGER FUNCTION PTISON (I, NODE)

Mnemonic :
   Parse tree node ith son.
Input Parameters :
   I(integer) - the index of the node desired
                (I.GE.1) and (I.LE.PTNMSN (NODE))
   NODE(integer) - a parse tree node.
Result :
   The parse tree node that is the I'th son of node, NODE.
   If I is not it the correct range or NODE is not a valid node,
   -1 is returned.


INTEGER FUNCTION PTNDTP (NODE)

Mnemonic :
   Parse tree node type.
Input Parameters :
   NODE(integer) - a parse tree node.
Result :
   The type (name) of the parse tree node.
   If NODE is not a valid node, -1 is returned.


LOGICAL FUNCTION PTISSM (NODE)

Mnemonic :
   Parse tree node has a symbol.
Input Parameters :
   NODE(integer) - a parse tree node.
Result :
   TRUE iff the node, NODE, has an associated symbol.
   (Can only be true for leaves.)
   If NODE is not a valid node, .FALSE. is returned.


INTEGER FUNCTION PTSYMB (NODE)

Mnemonic :
   Parse tree node symbol.
Input Parameters :
```

```
   NODE(integer) - a parse tree node.
Result :
   The symbol of the node, NODE.
   If NODE is not a valid node or PTISSM (NODE) returns .FALSE.,
   -1 is returned.


INTEGER FUNCTION SYMLEN (SYMBOL)

Mnemonic :
   Get the length of a symbol.
Input Parameters :
   SYMBOL(integer) - a symbol.
Result :
   Length of the symbol, SYMBOL (number of characters).
   If SYMBOL is not a valid symbol, -1 is returned.


INTEGER FUNCTION SYMCHR (SYMBOL, I)

Mnemonic :
   Get the Ith character of a symbol.
Input Parameters:
   SYMBOL(integer) - a symbol.
   I(integer) - the index of the character desired.
                (I.GE.1) and (I.LE.SYMLEN (SYMBOL))
Result :
   The Ith character of the symbol, SYMBOL,
   stored in A1 format.
   If I is not in the correct range or SYMBOL is not a valid symbol,
   -1 is returned.
```

References

[1] Geoffrey M. Clemm, FSCAN83 Report and User's Manual, Univ. of Colorado Tech. Report, #CU-CS-248-83, June, 1983.

[2] Charles Wetherell and Alfred Shannon, "LR Automatic Parser Generator and LR(1) Parser" IEEE Transactions on Software Engineering, Vol SE-7#3, May 1981, p.274.

[3] Geoffrey M. Clemm, FSCAN83 Report and User's Manual, Univ. of Colorado Tech. Report, #CU-CS-248-83.

[4] Stephen C. Johnson, "YACC - Yet Another Compiler-Compiler", Bell Lab. Computing Science Tech. Rept. #32, July 1975.

Appendix A:
Machine Dependencies in the TREGRM compiler

1. Machine Dependent Constants

1.1. NBTPWD

NBTPWD in /NBTPWC/ is the number of bits in a machine word.

2. Machine Dependent Primitives

2.1. INTEGER FUNCTION INTGER (CHAR)

Input:
CHAR contains a character stored in 1H (or A1) format.

Result:
The ASCII code for the character, CHAR (an integer between 0 and 127).

2.2. INTEGER FUNCTION CHRCTR (INT)

This is the inverse of the INTGER function.

2.3. INTEGER FUNCTION DIG (CHAR)

Input:
same as INTGER

Result:
If the character is a digit the result is the integer value of the digit (0-9); otherwise the result is -1.

2.4. INTEGER FUNCTION IAND (I1,I2)
INTEGER FUNCTION IOR (I1,I2)
INTEGER FUNCTION INOT (I1)

These functions return the result of the bitwise logical operation of AND, OR and NOT, respectively.

2.5. INTEGER FUNCTION HOLCHR (HCONST,ICHAR)

Input:
HCONST is a Hollerith constant of the form nHc_1c_2...c_n where n is an unsigned positive integer and c_i is a character, i=1..n. ICHAR is an integer between 1 and n.

Result:
HOLCHR(HCONST,i) will return c_i, stored in A1 or 1H format.

2.6. INTEGER FUNCTION LRS (IVAL, ICOUNT)
INTEGER FUNCTION LLS (IVAL, ICOUNT)

LRS and LLS return the logical shift (end-off, zero-fill), right and left respectively, of ICOUNT binary positions of the value, IVAL.

Appendix B:
Machine Dependencies in the TREGRM Standard Driver.

The following machine dependent primitives are required:

1. INTEGER FUNCTION INTGER (CHAR)

2. INTEGER FUNCTION CHRCTR (INT)

3. INTEGER FUNCTION DIG (CHAR)

3. INTEGER FUNCTION HOLCHR (HCONST, ICHAR)

4. INTEGER FUNCTION LRS (IVAL, ICOUNT)

5. INTEGER FUNCTION LLS (IVAL, ICOUNT)

These routines are described in Appendix A.

Appendix C :
Examples of TREGRM Programs

Following are two complete TREGRM programs. They describe syntactic analyzers for the TREGRM language and FORTRAN-77 respectively.

```
# THIS IS THE TREGRM PROGRAM USED TO CREATE THE PARSE TREE
# GENERATOR FOR THE TREGRM COMPILER.

SCANNER
  "NAME"    = IDNTFR => "NAME"
  "DSTRNG" = DSTRNG => "DSTRNG"
  "KSTRNG" = KSTRNG => "KSTRNG"

SCREENER
  ('SCANNER' 'SCREENER' 'NODES' 'RULES'
   '=' '=>' '->' '+' '//' '?' ';' '(' ')')
  = (IDNTFR OPRATR DELMTR)

NODES
  "PROGRM" "SCANNR" "NONLDF" "LEAFDF" "SCRNR"
  "KEYWDS" "STOKNS" "NODES"  "RULES"  "RULE"
  "ALTLST" "ALTRNT" "SEQ"    "LIST"   "PLUS"
  "OPTNAL" "OUTNOD" "NAME"   "DSTRNG" "KSTRNG"

RULES
TREGRM
   -> SCANR SCRNR NODS RULS        => "PROGRM"
   -> SCANR NODS RULS              => "PROGRM" ;
SCANR
   -> 'SCANNER' LEXDEFN+           => "SCANNR" ;
LEXDEFN
   -> "DSTRNG" '=' "NAME"          => "NONLDF"
   -> "KSTRNG" '=' "NAME" '=>' "KSTRNG"
                                   => "LEAFDF" ;
SCRNR
   -> 'SCREENER' SCRNKWDS '=' SCRNTKNS
                                   => "SCRNR" ;
SCRNKWDS
   -> '(' "DSTRNG"+ ')'            => "KEYWDS" ;
SCRNTKNS
   -> '(' "NAME"+ ')'              => "STOKNS" ;
NODS
   -> 'NODES' "KSTRNG"+            => "NODES" ;
RULS
   -> 'RULES' RULE+                => "RULES" ;
RULE
   -> "NAME" ALTLST ';'            => "RULE" ;
ALTLST
   -> ALTRNT+                      => "ALTLST" ;
ALTRNT
   -> '->' EXPR                    => "ALTRNT"
   -> '->' EXPR TREESPEC           => "ALTRNT" ;
EXPR
   -> TERM+                        => "SEQ"? ;
TERM
   -> ELMNT '//' ELMNT             => "LIST"
   -> ELMNT '+'                    => "PLUS"
   -> ELMNT ;
```

```
ELMNT
    -> '(' EXPR ')'
    -> "NAME"
    -> "KSTRNG"
    -> "DSTRNG" ;
TREESPEC
    -> '=>' NODNAM
    -> '=>' NODNAM '?'                  => "OPTNAL"
    -> '=>' '(' NODNAM ')'              => "OUTNOD" ;
NODNAM
    -> "KSTRNG" ;
```

```
SCANNER
  "INTCNST"      = DCONST   => "ICONST"
  "NAME"         = NAME     => "NAME"
  "LGCLCNST"     = LCONST   => "LCONST"
  "REALCNST"     = RCONST   => "RCONST"
  "DBLPCNST"     = DPCNST   => "DPCNST"
  "EDITDSC"      = FIELD    => "FMTFLD"
  "STRCNST"      = SCONST   => "SCONST"
  "HOLCNST"      = HCONST   => "HCONST"
  'LPAREN'       = LPAREN
  'RPAREN'       = RPAREN
  'EQV'          = EQV
  'NEQV'         = NEQV
  'OR'           = OR
  'AND'          = AND
  'NOT'          = NOT
  'LT'           = LT
  'LE'           = LE
  'EQ'           = EQ
  'NE'           = NE
  'GT'           = GT
  'GE'           = GE
  'DBLSLASH'     = CONCAT
  'DBLSTAR'      = DBASTR
  'STAR'         = ASTRSK
  'SLASH'        = SLASH
  'PLUS'         = PLUS
  'MINUS'        = MINUS
  'COMMA'        = COMMA
  'EQUALS'       = EQUALS
  'COLON'        = COLON
  'EOS'          = EOS
  'END'          = KEND
  'PROGRAM'      = KPROGR
  'FUNCTION'     = KFUNCT
  'INTEGER'      = KINTEG
  'REAL'         = KREAL
  'DOUBLE'       = KDOUBL
  'PRECISION'    = KPRECI
  'COMPLEX'      = KCOMPL
  'LOGICAL'      = KLOGIC
  'CHARACTER'    = KCHARA
  'SUBROUTINE'   = KSUBRO
  'ENTRY'        = KENTRY
  'BLOCK'        = KBLOCK
  'DATA'         = KDATA
  'DIMENSION'    = KDIMEN
  'COMMON'       = KCOMMO
  'IMPLICIT'     = KIMPLI
  'PARAMETER'    = KPARAM
  'EXTERNAL'     = KEXTER
  'INTRINSIC'    = KINTRI
  'SAVE'         = KSAVE
```

```
   'ASSIGN'      = KASSIG
   'GO'          = KGO
   'TO'          = KTO
   'IF'          = KIF
   'THEN'        = KTHEN
   'ELSE'        = KELSE
   'DO'          = KDO
   'CONTINUE'    = KCONTI
   'STOP'        = KSTOP
   'PAUSE'       = KPAUSE
   'WRITE'       = KWRITE
   'READ'        = KREAD
   'PRINT'       = KPRINT
   'OPEN'        = KOPEN
   'CLOSE'       = KCLOSE
   'INQUIRE'     = KINQUI
   'BACKSPACE'   = KBACKS
   'ENDFILE'     = KENDFI
   'REWIND'      = KREWIN
   'FORMAT'      = KFORMA
   'CALL'        = KCALL
   'RETURN'      = KRETUR
   'EQUIVALENC'  = KEQUIV

NODES
   "F77PRG"  "PRGUNT"  "LABLD"   "END"     "PROG"    "FUNC"    "INTGR"
   "REAL"    "DBLPRC"  "COMPLX"  "LOGICL"  "CHRCTR"  "LIST"    "SUBR"
   "ASTRSK"  "ENTRY"   "BLKDTA"  "DIMNSN"  "ARDCL"   "ARDIMS"  "ARDIM"
   "DARDIM"  "EQVLNC"  "EQVSET"  "COMMON"  "BLNKCM"  "LBLDCM"  "CBITMS"
   "TYPE"    "DCLITS"  "CHRLEN"  "IMPLCT"  "IMPDCL"  "CHRRNG"  "PARMTR"
   "PRMDCL"  "EXTRNL"  "INTRN"   "SAVE"    "CBLKNM"  "DATA"    "DTADCL"
   "DTAITS"  "DTAVLS"  "MULTDV"  "NEG"     "DIDLST"  "DOSPEC"  "ASGN"
   "ASSIGN"  "ASORSF"  "AOSDEF"  "GOTO"    "CMGOTO"  "ASGOTO"  "LBLLST"
   "ARTHIF"  "AILBLS"  "LOGIF"   "IFTHEN"  "ELSEIF"  "ELSE"    "ENDIF"
   "DO"      "CNTNU"   "STOP"    "PAUSE"   "WRITE"   "READ"    "PRINT"
   "CILIST"  "EQUALS"  "CONCAT"  "IOIMDL"  "OPEN"    "CLOSE"   "INQUIR"
   "BCKSPC"  "ENDFIL"  "REWIND"  "FORMAT"  "REPEAT"  "SLASH"   "COLON"
   "CALL"    "LBLARG"  "RETURN"  "EQV"     "NEQV"    "OR"      "AND"
   "NOT"     "LT"      "LE"      "EQ"      "NE"      "GT"      "GE"
   "PLUS"    "MINUS"   "POS"     "MLTPLY"  "DIVIDE"  "EXPONT"  "SPAREN"
   "CCONST"  "SUBSTR"  "AOFREF"  "ARGLST"  "ARELM"   "SSSPEC"  "DEFALT"
   "ICONST"  "NAME"    "LCONST"  "RCONST"  "DPCNST"  "FMTFLD"  "HCONST"
   "SCONST"  "LABEL"

RULES
F77PRG      -> PRGUNIT+                                  => "F77PRG" ;
PRGUNIT     -> BODYSTMT+ ENDSTMT                         => "PRGUNT"
            -> ENDSTMT                                   => "PRGUNT" ;
BODYSTMT    -> LABEL STMT 'EOS'                          => "LABLD"
            -> STMT 'EOS'
            -> LABEL FORMAT 'EOS'                        => "LABLD" ;
ENDSTMT     -> LABEL END 'EOS'                           => "LABLD"
            -> END 'EOS' ;
```

```
END         -> 'END'                                    => "END" ;
STMT        -> PROG
            -> FUNC
            -> SUBR
            -> BLKDTA
            -> ENTRY
            -> PARM
            -> IMPL
            -> DATA
            -> DIM
            -> EQUIV
            -> COMMON
            -> TYPE
            -> EXTRNL
            -> INTRNSC
            -> SAVE
            -> DO
            -> LOGIF
            -> IFTHEN
            -> ELSIF
            -> ELSE
            -> ENDIF
            -> ASGN
            -> ASGNORSF
            -> GOTO
            -> ARTHIF
            -> CNTNU
            -> STOP
            -> PAUSE
            -> READ
            -> WRITE
            -> PRINT
            -> RWND
            -> BKSPC
            -> ENDFIL
            -> OPEN
            -> CLOSE
            -> INQUIRE
            -> CALL
            -> RETURN ;
PROG        -> 'PROGRAM' "NAME"                         => "PROG" ;
FUNC        -> FUNCPREFIX "NAME" FPLIST                 => "FUNC" ;
FUNCPREFIX  -> TYP 'FUNCTION'
            -> 'FUNCTION'                               => "DEFALT" ;
TYP         -> 'INTEGER'                                => "INTGR"
            -> 'REAL'                                   => "REAL"
            -> 'DOUBLE' 'PRECISION'                     => "DBLPRC"
            -> 'COMPLEX'                                => "COMPLX"
            -> 'LOGICAL'                                => "LOGICL"
            -> 'CHARACTER'                              => "CHRCTR"
            -> 'CHARACTER' 'STAR' LENSPEC               => "CHRCTR" ;
FPLIST      -> 'LPAREN' "NAME"//'COMMA' 'RPAREN'        => "LIST"
            -> 'LPAREN' 'RPAREN' ;
```

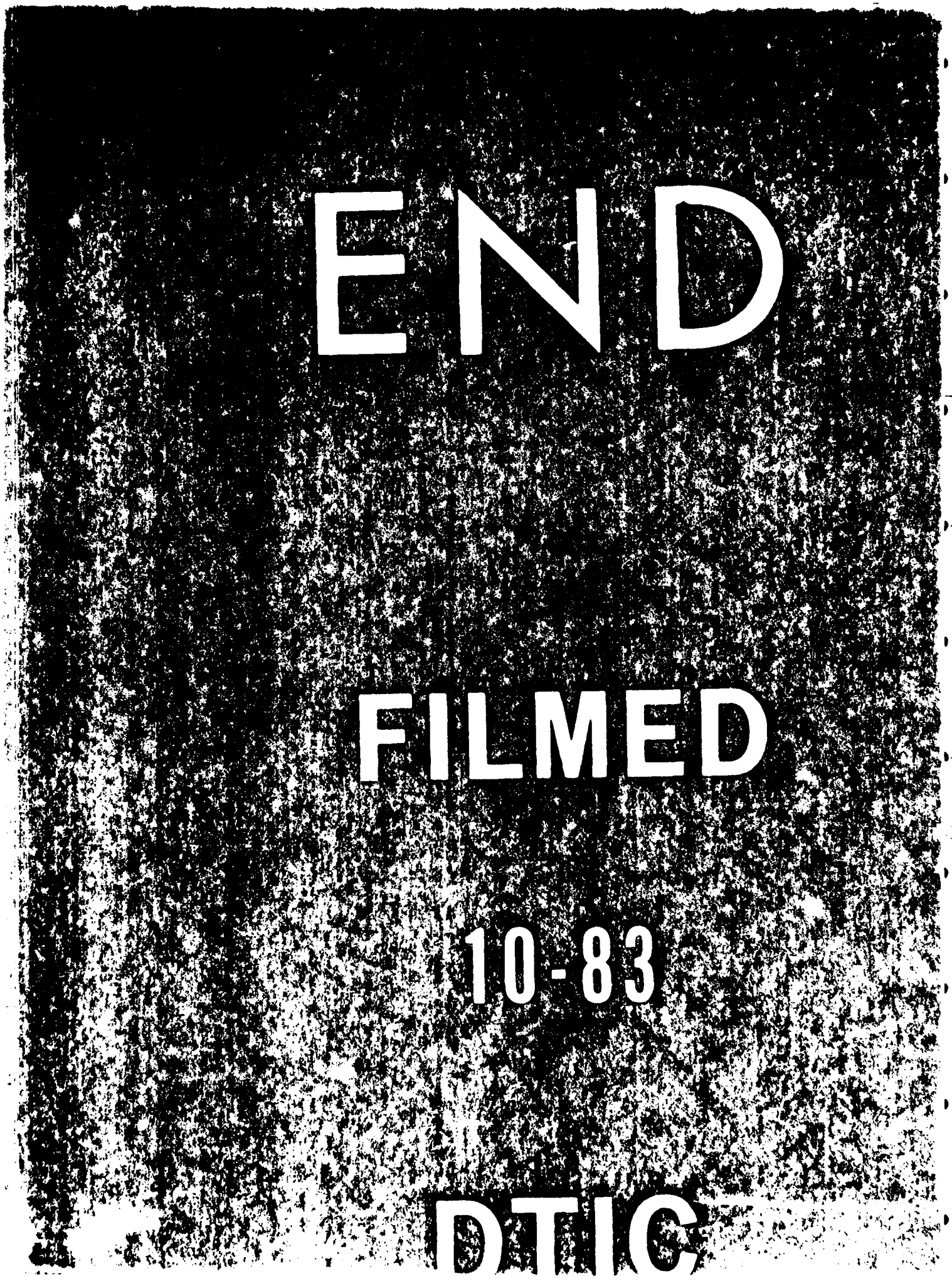
```
SUBR        -> 'SUBROUTINE' "NAME"                          => "SUBR"
            -> 'SUBROUTINE' "NAME" SPLIST                   => "SUBR" ;
SPLIST      -> 'LPAREN' 'RPAREN'
            -> 'LPAREN' SPARM//'COMMA' 'RPAREN'             => "LIST" ;
SPARM       -> "NAME"
            -> 'STAR'                                       => "ASTRSK" ;
ENTRY       -> 'ENTRY' "NAME"                               => "ENTRY"
            -> 'ENTRY' "NAME" SPLIST                        => "ENTRY" ;
BLKDTA      -> 'BLOCK' 'DATA'                               => "BLKDTA"
            -> 'BLOCK' 'DATA' "NAME"                        => "BLKDTA" ;
DIM         -> 'DIMENSION' ARDCL//'COMMA'                   => "DIMNSN" ;
ARDCL       -> "NAME" ARDIMLST                              => "ARDCL";
ARDIMLST    -> 'LPAREN' ARDIMS 'RPAREN'                     => "ARDIMS" ;
ARDIMS      -> ARDIM 'COMMA' ARDIMS
            -> ARDIM
            -> DARDIM ;
ARDIM       -> DIMBD 'COLON' DIMBD                          => "ARDIM"
            -> DIMBD                                        => "ARDIM" ;
DARDIM      -> DIMBD 'COLON' 'STAR'                         => "DARDIM"
            -> 'STAR'                                       => "DARDIM" ;
DIMBD       -> AEXPR ;
EQUIV       -> 'EQUIVALENC' EQVSET//'COMMA'                 => "EQVLNC" ;
EQVSET      -> 'LPAREN' EQVENT//'COMMA' 'RPAREN'            => "EQVSET" ;
EQVENT      -> "NAME"
            -> ARELM
            -> SUBSTR ;
COMMON      -> 'COMMON' CBLKLST                             => "COMMON"
            -> 'COMMON' BCILST CBLKLST                      => "COMMON"
            -> 'COMMON' BCILST                              => "COMMON" ;
CBLKLST     -> CBLK+ ;
CBLK        -> 'DBLSLASH' CBILST                            => "BLNKCM"
            -> 'SLASH' "NAME" 'SLASH' CBILST                => "LBLDCM" ;
BCILST      -> CBILST                                       => "BLNKCM" ;
CBILST      -> CBITMS                                       => "CBITMS" ;
CBITMS      -> CBITM 'COMMA' CBITMS
            -> CBITM 'COMMA'
            -> CBITM ;
CBITM       -> "NAME"
            -> ARDCL ;
TYPE        -> TYP DCLITS                                   => "TYPE"
            -> CHRTYP 'COMMA' DCLITS                        => "TYPE" ;
CHRTYP      -> 'CHARACTER' 'STAR' LENSPEC                   => "CHRCTR" ;
DCLITS      -> DCLITM//'COMMA'                              => "DCLITS" ;
DCLITM      -> DCLVAR
            -> DCLVAR 'STAR' LENSPEC                        => "CHRLEN" ;
DCLVAR      -> "NAME"
            -> ARDCL ;
IMPL        -> 'IMPLICIT' IMPDCL//'COMMA'                   => "IMPLCT" ;
IMPDCL      -> TYP 'LPAREN' CHRRNG//'COMMA' 'RPAREN'
                                                            => "IMPDCL" ;
CHRRNG      -> CHAR 'MINUS' CHAR                            => "CHRRNG"
            -> CHAR ;
CHAR        -> "NAME" ;
```

```
LENSPEC     -> 'LPAREN' 'STAR' 'RPAREN'                 => "ASTRSK"
            -> "INTCNST"
            -> 'LPAREN' AEXPR 'RPAREN' ;
PARM        -> 'PARAMETER' 'LPAREN' PRMDCL//'COMMA' 'RPAREN'
                                                        => "PARMTR" ;
PRMDCL      -> "NAME" 'EQUALS' EXPR                     => "PRMDCL" ;
EXTRNL      -> 'EXTERNAL' "NAME"//'COMMA'               => "EXTRNL" ;
INTRNSC     -> 'INTRINSIC' "NAME"//'COMMA'              => "INTRN" ;
SAVE        -> 'SAVE'                                   => "SAVE"
            -> 'SAVE' SAVITM//'COMMA'                   => "SAVE" ;
SAVITM      -> "NAME"
            -> 'SLASH' "NAME" 'SLASH'                   => "CBLKNM" ;
DATA        -> 'DATA' DTALST                            => "DATA" ;
DTALST      -> DTADCL
            -> DTALST DTADCL
            -> DTALST 'COMMA' DTADCL ;
DTADCL      -> DTAITS 'SLASH' DTAVLS 'SLASH'            => "DTADCL" ;
DTAITS      -> DTAITM//'COMMA'                          => "DTAITS" ;
DTAVLS      -> DTAVAL//'COMMA'                          => "DTAVLS" ;
DTAITM      -> VAR
            -> DIDLST ;
VAR         -> "NAME"
            -> ARELM
            -> SUBSTR ;
DTAVAL      -> DVCOUNT 'STAR' DVVAL                     => "MULTDV"
            -> DVVAL ;
DVCOUNT     -> "INTCNST"
            -> "NAME" ;
DVVAL       -> CONST
            -> 'PLUS' ARTHCNST
            -> 'MINUS' ARTHCNST                         => "NEG"
            -> "NAME" ;
DIDLST      -> 'LPAREN' DIDITS 'COMMA' DOSPEC 'RPAREN'
                                                        => "DIDLST" ;
DIDITS      -> DIDITM//'COMMA' ;
DIDITM      -> ARELM
            -> DIDLST ;
DOSPEC      -> "NAME" 'EQUALS' AEXPR 'COMMA' AEXPR
                                                        => "DOSPEC"
            -> "NAME" 'EQUALS' AEXPR 'COMMA' AEXPR 'COMMA' AEXPR
                                                        => "DOSPEC" ;
ASGN        -> "NAME" 'EQUALS' EXPR                     => "ASGN"
            -> SUBSTR 'EQUALS' FACTOR                   => "ASGN"
            -> 'ASSIGN' LABEL 'TO' "NAME"               => "ASSIGN" ;
ASGNORSF    -> ARORSFD 'EQUALS' EXPR                    => "ASORSF" ;
ARORSFD     -> "NAME" 'LPAREN' ARGLIST 'RPAREN'         => "AOSDEF" ;
GOTO        -> 'GO' 'TO' LABEL                          => "GOTO"
            -> 'GO' 'TO' 'LPAREN' LBLLST 'RPAREN' AEXPR
                                                        => "CMGOTO"
            -> 'GO' 'TO' 'LPAREN' LBLLST 'RPAREN' 'COMMA' AEXPR
                                                        => "CMGOTO"
            -> 'GO' 'TO' "NAME" 'LPAREN' LBLLST 'RPAREN'
                                                        => "ASGOTO"
```

```
                                                    => "ASGOTO"
            -> 'GO' 'TO' "NAME" 'COMMA' 'LPAREN' LBLLST 'RPAREN'
                                                    => "ASGOTO" ;
LBLLST      -> LABEL//'COMMA'                       => "LBLLST" ;
ARTHIF      -> 'IF' 'LPAREN' EXPR 'RPAREN' ARIFLABELS
                                                    => "ARTHIF" ;
ARIFLABELS  -> LABEL 'COMMA' LABEL 'COMMA' LABEL    => "AILBLS" ;
LOGIF       -> 'IF' 'LPAREN' EXPR 'RPAREN' STMT     => "LOGIF" ;
IFTHEN      -> 'IF' 'LPAREN' EXPR 'RPAREN' 'THEN'   => "IFTHEN" ;
ELSIF       -> 'ELSE' 'IF' 'LPAREN' EXPR 'RPAREN' 'THEN'
                                                    => "ELSEIF" ;
ELSE        -> 'ELSE'                               => "ELSE" ;
ENDIF       -> 'END' 'IF'                           => "ENDIF" ;
DO          -> 'DO' LABEL 'COMMA' DOSPEC            => "DO"
            -> 'DO' LABEL DOSPEC                    => "DO" ;
CNTNU       -> 'CONTINUE'                           => "CNTNU" ;
STOP        -> 'STOP'                               => "STOP"
            -> 'STOP' SPVAL                         => "STOP" ;
PAUSE       -> 'PAUSE'                              => "PAUSE"
            -> 'PAUSE' SPVAL                        => "PAUSE" ;
SPVAL       -> "INTCNST"
            -> "STRCNST" ;
WRITE       -> 'WRITE' CILIST                       => "WRITE"
            -> 'WRITE' CILIST OUTPUTLIST            => "WRITE" ;
READ        -> 'READ' XFMTID                        => "READ"
            -> 'READ' 'LPAREN' CIITEM 'RPAREN'      => "READ"
            -> 'READ' XCILIST                       => "READ"
            -> 'READ' XFMTID 'COMMA' INPUTLIST      => "READ"
            -> 'READ' 'LPAREN' CIITEM 'RPAREN' 'COMMA' INPUTLIST
                                                    => "READ"
            -> 'READ' CILIST INPUTLIST              => "READ" ;
PRINT       -> 'PRINT' CIITEM                       => "PRINT"
            -> 'PRINT' CIITEM 'COMMA' OUTPUTLIST    => "PRINT" ;
CILIST      -> 'LPAREN' CIITEM 'RPAREN'             => "CILIST"
            -> XCILIST ;
CIESPEC     -> "NAME" 'EQUALS' CIITEM               => "EQUALS" ;
CIITEM      -> EXPR
            -> ASTRSK ;
ASTRSK      -> 'STAR'                               => "ASTRSK" ;
XFMTID      -> ASTRSK
            -> "INTCNST"
            -> "STRCNST"
            -> "NAME"
            -> AOFREF
            -> SUBSTR
            -> SXFMTID 'DBLSLASH' EXPR              => "CONCAT"
# CIITEM USED TO PREVENT REDUCE CONFLICT
            -> 'LPAREN' CIITEM 'RPAREN' 'DBLSLASH' EXPR
                                                    => "CONCAT" ;
SXFMTID     -> "STRCNST"
            -> "NAME"
            -> AOFREF
            -> SUBSTR ;
```

```
XCILIST     -> 'LPAREN' CIESPECS 'RPAREN'              => "CILIST"
            -> 'LPAREN' CIITEM 'COMMA' CIESPECS 'RPAREN'
                                                       => "CILIST"
            -> 'LPAREN' CIITEM 'COMMA' CIITEM 'RPAREN'
                                                       => "CILIST"
            -> 'LPAREN' CIITEM 'COMMA' CIITEM 'COMMA' CIESPECS 'RPAREN'
                                                       => "CILIST" ;
CIESPECS    -> CIESPEC//'COMMA' ;
OUTPUTLIST  -> OUTPUTITEM//'COMMA' ;
# SEE COMMENT FOR COMPLEX CONSTANT
OUTPUTITEM  -> EXPR
            -> 'LPAREN' OUTPUTLIST 'COMMA' DOSPEC 'RPAREN'
                                                       => "IOIMDL" ;
INPUTLIST     -> INPUTITEM//'COMMA' ;
INPUTITEM     -> VAR
            -> 'LPAREN' INPUTLIST 'COMMA' DOSPEC 'RPAREN'
                                                       => "IOIMDL" ;
OPEN        -> 'OPEN' CILIST                           => "OPEN" ;
CLOSE       -> 'CLOSE' CILIST                          => "CLOSE" ;
INQUIRE     -> 'INQUIRE' 'LPAREN' INQSPECS 'RPAREN'
                                                       => "INQUIR"
            -> 'INQUIRE' 'LPAREN' CIITEM 'RPAREN'      => "INQUIR"
            -> 'INQUIRE' 'LPAREN' CIITEM 'COMMA' INQSPECS 'RPAREN'
                                                       => "INQUIR" ;
INQSPECS    -> INQSPEC//'COMMA' ;
INQSPEC     -> "NAME" 'EQUALS' EXPR                    => "EQUALS" ;
BKSPC       -> 'BACKSPACE' FSPEC                       => "BCKSPC" ;
ENDFIL      -> 'ENDFILE' FSPEC                         => "ENDFIL" ;
RWND        -> 'REWIND' FSPEC                          => "REWIND" ;
FSPEC       -> CIITEM
             -> 'LPAREN' CIESPECS 'RPAREN'             => "CILIST" ;
FORMAT      -> 'FORMAT' FRMT
            -> 'FORMAT' 'LPAREN' 'RPAREN'              => "FORMAT" ;
FRMT        -> 'LPAREN' FMTITEM+ 'RPAREN'              => "FORMAT" ;
FMTITEM     -> "EDITDSC"
            -> "STRCNST"
            -> "HOLCNST"
            -> FRMT
            -> "INTCNST" FRMT                          => "REPEAT"
            -> 'COMMA'
            -> 'SLASH'                                 => "SLASH"
            -> 'COLON'                                 => "COLON" ;
CALL        -> 'CALL' "NAME"                           => "CALL"
            -> 'CALL' "NAME" 'LPAREN' 'RPAREN'         => "CALL"
            -> 'CALL' "NAME" 'LPAREN' ARG//'COMMA' 'RPAREN'
                                                       => "CALL" ;
ARG         -> EXPR
            -> 'STAR' LABEL                            => "LBLARG" ;
RETURN      -> 'RETURN'                                => "RETURN"
            -> 'RETURN' AEXPR                          => "RETURN" ;
EXPR        -> EXPR 'EQV' LOGEXPR                      => "EQV"
            -> EXPR 'NEQV' LOGEXPR                     => "NEQV"
            -> LOGEXPR ;
```

```
LOGEXPR     -> LOGEXPR 'OR' LOGTERM                  => "OR"
            -> LOGTERM ;
LOGTERM     -> LOGTERM 'AND' LOGFACTOR               => "AND"
            -> LOGFACTOR ;
LOGFACTOR   -> 'NOT' LOGPRIM                         => "NOT"
            -> LOGPRIM ;
LOGPRIM     -> AEXPR 'LT' AEXPR                      => "LT"
            -> AEXPR 'LE' AEXPR                      => "LE"
            -> AEXPR 'EQ' AEXPR                      => "EQ"
            -> AEXPR 'NE' AEXPR                      => "NE"
            -> AEXPR 'GT' AEXPR                      => "GT"
            -> AEXPR 'GE' AEXPR                      => "GE"
            -> AEXPR ;
AEXPR       -> AEXPR 'PLUS' ATERM                    => "PLUS"
            -> AEXPR 'MINUS' ATERM                   => "MINUS"
            -> 'PLUS' ATERM                          => "POS"
            -> 'MINUS' ATERM                         => "NEG"
            -> ATERM ;
ATERM       -> ATERM 'STAR' FACTOR                   => "MLTPLY"
            -> ATERM 'SLASH' FACTOR                  => "DIVIDE"
            -> FACTOR ;
FACTOR      -> PRIMARY 'DBLSTAR' FACTOR              => "EXPONT"
            -> PRIMARY 'DBLSLASH' FACTOR             => "CONCAT"
            -> PRIMARY ;
PRIMARY     -> CONST
            -> "NAME"
            -> AOFREF
            -> SUBSTR
            -> 'LPAREN' EXPR 'RPAREN'                => "SPAREN" ;
CONST       -> ARTHCNST
            -> "STRCNST"
            -> "HOLCNST"
            -> "LGCLCNST" ;
ARTHCNST    -> "INTCNST"
            -> "REALCNST"
            -> "DBLPCNST"
# PRINT 5,(3,2) ... PRINT 5,(3,2,I=1,10)
            -> 'LPAREN' OUTPUTLIST 'COMMA' ARTHCNST 'RPAREN'
                                                     => "CCONST" ;
SUBSTR      -> ARELM SSSPEC                          => "SUBSTR"
            -> "NAME" SSSPEC                         => "SUBSTR" ;
AOFREF      -> "NAME" 'LPAREN' ARGLIST 'RPAREN'      => "AOFREF" ;
ARGLIST     -> EXPR//'COMMA'                         => "ARGLST" ;
ARELM       -> "NAME" 'LPAREN' ARGLIST 'RPAREN'      => "ARELM" ;
SSSPEC      -> LSSSPEC 'COLON' RSSSPEC               => "SSSPEC" ;
LSSSPEC     -> 'LPAREN' AEXPR
            -> 'LPAREN'                              => "DEFALT" ;
RSSSPEC     -> AEXPR 'RPAREN'
            -> 'RPAREN'                              => "DEFALT" ;
LABEL       -> "INTCNST"                             => "LABEL" ;
```



END

FILMED

10-83

DTIC